

RECEIVED
CENTRAL FAX CENTER

NOV 09 2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCESIn re Application of:
Broussard

Serial No. 09/870,615

Filed: May 31, 2001

For: INHERITANCE OF BACKGROUND
COLOR IN A CONTAINMENT
HIERARCHY OF OBJECTS IN A
GRAPHICAL USER INTERFACEGroup Art Unit: 2173
Examiner: Bonshock, D.Atty. Dkt. No. AUS920010264US1
(5468-07700)

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313, on the date indicated below; *or transmitted via facsimile.*

11/09/2004
Date

Kevin L. Daffer

APPEAL BRIEFBox AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

Sir:

Further to the Notice of Appeal faxed to and received by the U.S. Patent and Trademark Office on September 14, 2004, Appellant presents this Appeal Brief. The Notice of Appeal was filed following mailing of a final Office Action on June 15, 2004. Appellant hereby appeals to the Board of Patent Appeals and Interferences from a final rejection of claims 1-6, 8-13, and 15-19 in the final Office Action, and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by International Business Machines Corporation, a corporation having its principal place of business at New Orchard Road, Armonk, New York, 10504, as evidenced by the assignment recorded at Reel 011888, Frame 0545.

II. RELATED APPEALS AND INTERFERENCES

Notices of Appeal have been filed for the following applications, which share a common specification with the application currently on appeal.

09/870,622 Notice of Appeal filed 8/24/04

09/870,621 Notice of Appeal filed 9/24/04

However, because dissimilar art is cited in the present application and the above-mentioned related applications, Appellants do not believe that the outcome of this appeal will have any bearing on the Board's decision on the related appeals. No other appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-15 were originally filed in the present application. Claims 1, 2, 4, 6, 8, 9, 11, 13, and 15 were amended, claims 16-19 were added, and claims 7 and 14 were canceled in a response filed April 14, 2004 to an Office Action mailed January 15, 2004. Claims 1-6, 8-13, and 15-19 stand finally rejected under 35 U.S.C. § 103, which are the subject of this appeal. A copy of claims 1-6, 8-13, and 15-19, as on appeal (incorporating entered amendments), is included in the Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims were filed subsequent to their final rejection. The Appendix hereto therefore reflects the current state of the claims.

V. SUMMARY OF THE INVENTION

Appellant's claimed invention relates to a system of software components (200, 202, 208, 210 and 212, Fig. 19B), a computer-readable storage device (18, Fig. 1) and a method (not shown) for establishing an order of background color inheritance between a parent object and a child object.

In some embodiments, a method for establishing the order of background color inheritance for a child object is provided herein. The method of color inheritance is performed in a particular order. First, if a background color of the child object is declared, the child object is displayed with the declared background color. Second, if the background color of the child object is not declared, but a background color is globally defined for the system of software components, the child object is displayed with the globally defined background color. The globally defined background color is independent of the operating system. Third, if the background color of the child object is not declared, and the background color is not globally defined for the system of software components, the child object is displayed with the background color of the parent object. (Specification -- page 41, lines 1-10).

In some embodiments, a system of software components is also provided for implementing the presently claimed order of color inheritance. For example, the system of software components may include a lightweight peer component and a lightweight proxy component for displaying the child object with either the globally defined or inherited background color, if the background color of the child object is not declared. In particular, the lightweight peer component may be adapted to redirect a method call for getting the background color to the lightweight proxy component, which in turn, may translate the method call to determine whether the child object should be displayed using the globally defined or inherited background color. In some embodiments, the lightweight proxy component may replace an AWT-based heavyweight peer component, so that a child object may inherit its background color from either an AWT-based or Swing-based parent object. (Specification -- page 24, line 15 to page 30, line 2, and page 41, lines 11-29).

In some embodiments, a computer-readable storage device is also provided for implementing the presently claimed order of color inheritance. For example, the computer-readable storage device may include a windows-based operating system and a system of software components within the operating system. The storage device may also include a parent object and a child object, both of which are created at runtime by an application running under the operating system. As noted above, the child object may be displayed by the system of software components in accordance with the presently claimed order of background color inheritance.

VI. ISSUES

1. Whether claims 1-6, 8-13, 15, and 16 are unpatentable under 35 U.S.C. § 103(a) over a web publication provided by MagcLang Institute, entitled *Swing Short Course, Part 1* (hereinafter "MagcLang") in view of a web publication provided by Sun Microsystems, entitled *Java Platform 1.2 Beta 4 API Specification: Class LookAndFeel* (hereinafter "JavaSPEC") in view of a web publication provided by JavaOne, entitled "Java Foundation Classes (a.k.a. "Swing") Component Architecture" (hereinafter "JavaOne"), and in further view of U.S. Patent No. 6,191,790 to Bogdan (hereinafter "Bogdan").
2. Whether claims 17-19 are unpatentable under 35 U.S.C. § 103(a) over MagcLang, JavaSPEC, JavaOne, Bogdan and a web publication, entitled *Mixing Heavy and Light Components*, by Amy Fowler (hereinafter "M-SUN").

VII. GROUPING OF CLAIMS

Claims 1-6, 8-13, 15 and 16 (Group I) stand or fall together.

Claims 17-19 (Group II) stand or fall together.

The reasons why the two groups of claims are believed to be separately patentable are explained below in the appropriate parts of the Argument.

VIII. ARGUMENT

In general, the specification provides a unique system of software components that enables an application program (e.g., a Java application) to be truly portable across all operating system (OS) platforms by providing various means for maintaining the "look and feel" of the application program. "A software program is said to be 'portable' across various platforms if the program can run without modification on any of those platforms." (Specification, page 6, lines 23-24). Before disclosing such means, the specification highlights several drawbacks of prior art attempts at application portability.

For example, Java application programs utilize a platform-dependent application program interface (API), commonly known as the Abstract Windowing Toolkit (AWT), to produce heavyweight software components that are written in native code (i.e., instructions specific to a particular OS). When

AWT heavyweight software components are used for displaying images, the "look and feel" of those images may differ depending on the particular OS running the Java application program. In the context of a graphical user interface (GUI), "the 'look and feel' of a GUI refers to such things as the appearance, color and behavior of Buttons, TextFields, Listboxes, menus, etc..." (Specification, page 7, line 4-8). Thus, there are certain limitations within the AWT that prohibit true portability of Java application programs, especially in terms of the look and feel of the application program. See, e.g., Specification, page 7, line 24 - page 9, line 10, and page 18, line 10 to page 19, line 23.

In an effort to overcome the platform-dependency of AWT-based programs, Swing was developed as part of the Java Foundation Classes (JFC). An API written using Swing contains no native code, and therefore, can be run on substantially any OS without changing the look and feel of the application. See, e.g., Specification, page 9, lines 14-30, and page 19, line 25 - page 20, line 4. Unfortunately, the lightweight software components of Swing cannot completely eliminate the platform-dependency of Java applications that use AWT. Since Swing versions of many AWT components (e.g., containers, such as frames) are unavailable, many programmers have attempted to mix Swing and AWT components within a given API. See, e.g., Specification, page 21, lines 1-13. However, straightforward mixing of Swing and AWT components tends to create many problems that programmers have simply learned to accept. See, e.g., Specification, page 22, lines 1-28.

One problem encountered when mixing Swing and AWT components within a given API arises with the issue of background color inheritance. In general, Swing and AWT-based applications employ substantially different background color inheritance schemes, which are often incompatible with one another. In AWT-based applications, for example, a child object will automatically inherit its background color from a parent object, if the child's background color is not explicitly set. This may be done, in some cases, by employing the "getBackground" method from the Java "Component" class. On the other hand, a child object in a Swing-based application can receive its background color assignment from globally-defined, OS-independent settings in Swing, if such settings are defined and the child's background color is not explicitly set. In some cases, globally-defined Swing settings may be set by employing the "installColorsAndFont" method from the Swing "LookAndFeel" class. However, a problem arises when one tries to impart the platform-independency of Swing to an AWT-based legacy application. In particular, heavyweight objects in the AWT-based legacy application cannot inherit their background color from globally-defined Swing settings, due to the inherent limitations of the color inheritance method (e.g., "getBackground" method) used by AWT-based legacy applications. In

addition, lightweight Swing objects generally cannot be used to directly replace the heavyweight AWT objects in an AWT-based legacy application, due to the additional problems this tends to create. *See, e.g., Specification, page 22, lines 1-28; and page 40, lines 8-22.*

Therefore, a need exists for a new method of color inheritance, which allows the background (and/or foreground) color of objects to be set by Swing, while also preserving the capability for objects to inherit this setting from their parent in legacy applications. *See, e.g., Specification, page 40, lines 22-25.*

The invention as recited in claims 1-6, 8-13 and 15-19 addresses the above-described need for a new color inheritance scheme by providing a system of software components – referred to as “AWTSwing” components – which enable a child object to be displayed with a background color that is either (i) explicitly declared for that child object, (ii) assigned by global “look and feel” Swing settings (which are independent of the operating system executing the software components), or (iii) inherited from a parent of the child object. The method for establishing the inheritance of background color for a child object is purposefully performed in the order presented above. In this manner, the new color inheritance scheme allows normal inheritance of background color from global “look and feel” Swing settings. However, the new color inheritance scheme defaults to the AWT scheme of inheriting background color from the parent object when global Swing settings are unavailable. An explicit declaration of the child object’s background color overrides either type of inheritance. As an additional advantage, the new color inheritance scheme offers considerable flexibility in displaying objects created by legacy Java applications. For example, when an object must inherit its background color from its parent (e.g., only when explicit and global settings are unavailable), either the parent of the original AWT-based object, or that of its Swing counterpart, can be used. This option is made possible by the replacement of AWT-based heavyweight Peers by AWTSwing proxy components. *See, e.g., Specification, page 41, lines 11-21 and Fig. 19B.*

As described in more detail below, none of the cited art, either separately or in combination, provides motivation to teach or suggest all steps of the presently claimed method of color inheritance, regardless of the order in which the steps are performed. Therefore, the teachings of the cited art cannot be used to render the limitations of the presently claimed case unpatentable.

ISSUE 1 ARGUMENTS**Patentability of Group I Claims 1-6, 8-13, 15 and 16**

1. **MageLang, JavaSPEC, JavaOne and Bogdan each fail to teach or suggest a system of software components (claim 1), a computer-readable storage device (claim 15) or a method (claim 8) for establishing color inheritance between a parent object and a child object in the order recited in claims 1, 8 and 15.**

Independent claim 8 states, in part:

A method for color inheritance between a parent object and a child object... wherein the method is adapted to display the child object in accordance with the following color inheritance order: if a background color of the child object is declared, displaying the child object with the declared background color; if the background color of the child object is not declared, and a background color is globally defined for the system of components, displaying the child object with the globally defined background color; and if the background color of the child object is not declared, and the background color setting is not globally defined for the system of components, displaying the child object with the background color of the parent.

Independent claims 1 (a system of software components) and 15 (a computer-readable storage device) recite the very same order of color inheritance. Thus, the specification and present claims provide a particular order by which a child object may inherit a background (and/or foreground) color. As noted above, the presently claimed order of color inheritance allows globally-defined, OS-independent Swing settings to be used for the background color of a child object before the background color is inherited from a parent object. Explicit declaration of a background color overrides all other means of color inheritance.

With regard to present claim 8, the Examiner suggests that "MageLang teaches a method of color inheritance (see page 7 under the JButton heading) between a system of components (see page 4)...to display a parent (parent window) and child object (the button) (see page 7 under the heading JButton)." (Office Action, pages 4-5). The Examiner further suggests that "MageLang also teaches that the background color of the JButton can be set directly, but if not set[, it's] default [color] is invisible (showing the parents color)." (Office Action, page 5). The Examiner admits that MageLang fails to teach "a middle step of checking for a globally defined color, and if defined, setting the background to the defined color" (Office Action, page 5). The Examiner provides similar suggestions for the presently claimed system of software components (claim 1) and the presently claimed computer-readable storage

device (claim 15) on pages 2-3 and 5-6 of the Office Action. Similar statements can also be found in the final Office Action.

In response to the Examiner's suggestions, the Applicant agrees that there is no teaching or suggestion within MageLang for the so-called "middle step" of present claims 1, 8 and 15, and further asserts that there is also no teaching or suggestion within MageLang for the last limitation of those claims. Support for the Applicant's assertions is provided below.

The MageLang reference provides a brief overview of Swing objects that may be included within a class of objects commonly known as the "JComponents" (See, e.g., MageLang, pages 1-35). Referring to page 4 of MageLang, the "JComboBox," "JLabel," "JButton," and "JPasswordField" objects are all Swing objects that descend from the JComponent class. Page 4 of MageLang also shows how the Swing objects of the JComponent class may be "child objects" of one or more "parent objects." For example, the "JButton" object is shown as a child object of the "AbstractButton" parent object, which in turn, is a child object of the "JComponent" class. As pointed out by the Examiner, these child objects may inherit their background color from a parent object, if a background color is not explicitly declared for a particular child object. In most cases, the background color can be explicitly declared for a child object by using the "setBackground" method from the Java "Component" class. The background color for the child object may then be obtained by using the "getBackground" method from the Java "Component" class. However, and as noted in the specification, the "getBackground" method "automatically returns the background color of the component's parent when there is no explicit declaration for the component." (Specification, page 42, lines 4-7). Since the "getBackground" method fails to determine whether or not a background color has been globally defined, the method cannot be used to teach or suggest any claim limitation that depends on the determination of a globally defined background color (e.g., the last and next to last limitations of claims 1, 8 and 15). As a consequence, the MageLang reference cannot be relied upon to teach or suggest the presently claimed method of color inheritance, as recited in claims 1, 8, and 15.

The Examiner suggests that teaching can be found within the JavaOne and JavaSPEC references for establishing a globally defined background color. For example, the Examiner states, "JavaOne teaches, on attached pages 7 and 8, building default tables to store default colors, fonts, icons and borders for components, and that all subsequent objects will use the new values." (Office Action, page 3, 5 and 6). The Examiner also states, "[i]t is further taught in JavaSPEC, on page 3, initializing the colors of the

background to the default color from the defaults table.” (Office Action, page 3, 5 and 6). The Applicant concedes that default colors, fonts, icons and borders may be stored in a Swing defaults table to establish the global “look and feel” settings for Swing child objects. The use of Swing default tables is commonly known in the art. However, the mere mention of Swing default tables does not and cannot provide the presently claimed method of color inheritance, as recited in claims 1, 8, and 15.

JavaOne and JavaSPEC both mention the Swing defaults table. For example, the JavaSPEC reference provides a brief overview of the Swing “LookAndFeel” class, including various methods for applying the look and feel settings that may be stored within a Swing defaults table (See, e.g., JavaSPEC, pages 1-7). In particular, the JavaSPEC reference indicates that the “installColorsAndFont” method of the “LookAndFeel” class may be used for initializing a component’s (e.g., a child object’s) foreground, background and font properties with the values obtained from a currently set defaults table. As such, the “installColorsAndFont” method may be used for applying a default background color to a child object. However, though the JavaOne and JavaSPEC references may disclose the existence of Swing default tables, they do not disclose the presently claimed method by which the default tables may be used. For example, the JavaSPEC reference does not teach or suggest that the “installColorsAndFont” method may enable a child object to inherit its background color from a parent object, if a background color is not explicitly or globally set for the child object. Therefore, the JavaOne and JavaSPEC references cannot be relied upon to teach or suggest the presently claimed method of color inheritance, as recited in claims 1, 8, and 15.

In the Office Action mailed January 15, 2004, the Examiner suggests that the “sequence of hierarchical inheritance is further taught by Bogdan, in column 1, line 65 through column 2, line 32, column 5, line 32, and column 7, lines 65 through column 8, line 14. He teaches inheritable properties, including background color, in which selection can be made as to which, either the parent or the global system setting... the child’s background color will be derived from.” (Office Action, pages 3 and 6). The Applicant respectfully disagrees with the Examiner’s suggestions, for at least the reasons set forth in more detail below.

Bogdan discloses a “system that supports a uniform three-dimensional rendering of components on a display screen by allowing a new component to inherit the shading properties of a parent component.” (Bogdan, Abstract). Bogdan, however, does not teach or suggest the presently claimed method of color inheritance. Instead, Bogdan provides a substantially different method of color

inheritance, as shown in Fig. 3 of Bogdan. The method of color inheritance, as disclosed by Bogdan, is as follows: "if a shading property is not defined in a given child component at the time the component is created, that child component can search its parent component hierarchy for inheritable shading properties. Absent any other source, shading properties and other properties can be obtained from available system-wide properties. However, using inherited properties ensures that each component in a component family hierarchy is consistent in its display characteristics." (Bogdan, column 2, lines 6-14, emphasis added; *see*, Fig. 3). Thus, the method of Bogdan differs from the presently claimed method of color inheritance, in one respect, by attempting to use inheritable shading properties (e.g., from a parent component) before "system-wide properties" may be used. As such, Bogdan fails to teach or suggest the presently claimed method of color inheritance, which employs globally-defined Swing settings before inheriting a background color from a parent object.

In the final Office Action mailed June 15, 2004, the Examiner admits "that Bogdan doesn't teach the specific order of operations as that of the presently claimed invention, however, claiming all the elements, but in a different order, is known to be a design feature and would be obvious over Bogdan." (Final Office Action, page 3). The Applicants appreciate the Examiner's recognition of Bogdan failing to teach the specific order of color inheritance taught in the presently claimed case. However, the Applicant's disagree with the Examiner's assertion that Bogdan discloses all elements of the presently claimed color inheritance scheme, regardless of the order in which they are presented. As described in more detail below, Bogdan does not disclose all elements. In addition, Applicant's wish to point out that, contrary to the statements submitted by the Examiner on page 9 of the final Office Action, the Applicants did not previously admit (in the Response to the Office Action) to Bogdan teaching all steps of the presently claimed method of color inheritance. The Applicants request that the erroneous statement be stricken from the record.

For example, and as argued in the Response mailed April 14, 2004, the "system-wide properties" of Bogdan are not independent of the operating system (*see*, e.g., Bogdan, column 1, lines 41-52). As such, the "system-wide properties" of Bogdan cannot be considered equivalent to a "globally defined background color," which is currently defined in present claims 1, 8, and 15 as independent of the operating system. For at least these reasons, Bogdan cannot be used to teach or suggest the "middle step" of the presently claimed order of background color inheritance. In other words, Bogdan does not teach or suggest that "if the background color of the child object is not declared, and a background color is

globally defined for the system of components, the child object is displayed with the globally defined background color, which is independent of the operating system,” as recited in present claims 1, 8, and 15. Therefore, Bogdan fails to teach or suggest “all elements” of the presently claimed method of color inheritance, in addition to “the specific order” in which the “elements” are performed.

For at least the reasons set forth above, MageLang, JavaSPEC, JavaOne, and Bogdan each fail to teach or suggest all limitations of present claims 1, 8, and 15.

2. **There is no motivation to combine or modify the teachings of MageLang, JavaSPEC, JavaOne and Bogdan to provide the presently claimed order of color inheritance.**

MageLang, JavaSPEC, JavaOne, and Bogdan each fail to provide motivation for modifying their teachings to provide the presently claimed order of color inheritance. For example, Bogdan fails to suggest a desirability for modifying the “system-wide properties” of Bogdan to be independent of the operating system running the user interface application. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01. Therefore, the “system-wide properties” of Bogdan cannot be modified to be independent of the operating system, since Bogdan fails to suggest a desirability for doing so.

Even if the “system-wide properties” of Bogdan were somehow modified to be independent of the operating system, the “system-wide properties” of Bogdan could not be used in the manner set forth in present claims 1, 8, and 15.

For example, Bogdan emphasizes “that only a root or top-level component will take its shading face color from the system property settings if no other color is expressly stated.” (Bogdan, column 7, lines 35-37). Since the “system-wide properties” of Bogdan are only applied to root or top-level components, the “system-wide properties” of Bogdan cannot be applied to child objects, as taught in the presently claimed case. Furthermore, the “system-wide properties” of Bogdan cannot be applied in the order set forth in the present claims, since Bogdan teaches away from using system-wide properties before using inheritable properties. As noted above, Bogdan utilizes system-wide properties only if a background color cannot be obtained from any other source (*See*, Bogdan, column 2, lines 6-14). Bogdan appears to consider the use of system-wide properties a disadvantage of prior art systems (*See*, e.g., Bogdan, column 1, lines 40-47). Therefore, even if the “system-wide properties” of Bogdan were

somehow modified to: (1) be independent of the operating system, and (2) be applied to child objects instead of top-level components, the teachings of Bogdan would still provide no motivation for using system-wide properties before using inheritable properties, because Bogdan specifically teaches away from doing so. It is noted that a *prima facie* case of obviousness may be rebutted by showing that the art, in any material respect, teaches away from the claimed invention. *In re Geisler*, 116 F.3d 1465, 1471, 43 USPQ2d 1362, 1366 (Fed. Cir. 1997); MPEP 2144.05 (III). As a consequence, Applicants strongly believe that the teachings of Bogdan cannot be relied upon to provide teaching or suggestion for the presently claimed method of color inheritance, as recited in claims 1, 8, and 15.

For at least the reasons set forth above, MageLang, JavaOne, JavaSPEC and Bogdan each fail to teach or suggest the presently claimed method of color inheritance. However, the Examiner suggests that "it would have been obvious to one of ordinary skill in the art, having the teachings of MageLang, JavaSPEC, JavaOne, and Bogdan before him at the time the invention was made to modify the JButton system of MageLang to include the use of a default table ... because default tables allow for a system to have specific settings, which make all windows uniform." (*See, e.g., Office Action, page 3, and Final Office Action, page 6*). As described below, the JButton system of MageLang, the default tables of JavaOne and JavaSPEC, and the inheritance sequence of Bogdan cannot be combined to provide the presently claimed method of color inheritance.

Applicants assert that it is improper to combine references where the references teach away from their combination. *In re Grasselli*, 713 F.2d 731, 743, 218 USPQ 769, 779 (Fed. Cir. 1983). MPEP 2145 (X)(D)(2). A *prima facie* case of obviousness may be rebutted by showing that the art, in any material respect, teaches away from the claimed invention. *In re Geisler*, 116 F.3d 1465, 1471, 43 USPQ2d 1362, 1366 (Fed. Cir. 1997); MPEP 2144.05 (III). Since Bogdan teaches away from the presently claimed method of color inheritance, the inheritance sequence of Bogdan cannot be used, alone or in combination with the remaining cited art, to establish a *prima facie* case of obviousness over the presently claimed case.

Furthermore, the remaining cited art cannot be combined or modified in such a manner that teaches or suggests the presently claimed order of color inheritance. For example, any methods of color inheritance that may be disclosed by MageLang, JavaOne and JavaSPEC are bound by predefined methodologies, or in other words, sequences of programming code that have been established and accepted as Swing methods for setting the background color. One simply cannot combine the methods of

Magelang, JavaOne and JavaSPEC without significantly modifying their normal programming code sequence. For example, one simply cannot combine the "setBackground" method (disclosed by Magelang) with the "installColorsAndFont" method (disclosed by JavaOne and JavaSPEC) to provide the presently claimed method of color inheritance without making any modifications to those methods.

In addition, Magelang, JavaOne and JavaSPEC provide absolutely no indication that these methods could be modified to provide the presently claimed method of color inheritance. For example, Magelang, JavaOne and JavaSPEC provide no indication or motivation for combining the "setBackground" and "installColorsAndFont" methods separately disclosed therein. Applicants assert that the teaching or suggestion to make the claimed combination must be found in the prior art, and not based on Applicant's own disclosure. *In re Vaack*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991); MPEP 2142. Therefore, Applicants assert that the Magelang, JavaOne and JavaSPEC references cannot be combined or modified to teach or suggest all limitations of present claims 1, 8, and 15.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the cited references can be found within the cited references to teach or suggest the aforementioned limitations of present claims 1, 8 and 15, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination, teaches all of the limitations of claim 1, 8 and 15, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-3 above, at least some limitations of claims 1, 8, and 15, as well as at least some limitations of claims 2-6, 9-13, and 15-19, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 1-6, 8-13, and 15-19 are patentably distinct over the cited art, and the rejection of Group I claims 1-6, 8-13, 15, and 16 under 35 U.S.C. § 103(a) is therefore asserted to be erroneous.

ISSUE 2 ARGUMENTS

Patentability of Group II Claims 17-19

Because claims 17-19 of Group II are dependent from claim 1 of Group I, the arguments presented above for patentability of claim 1 apply equally to claims 17-19, and are herein incorporated by reference. In addition to the arguments presented above with respect to claim 1, arguments are provided below to further establish the patentability of the current claims under 35 U.S.C. §103(a).

1. **Magelang, JavaSPEC, JavaOne, Bogdan and M-SUN fail to teach or suggest the presently claimed order of color inheritance and a system of software components, which can be invoked to display a child object in accordance with the presently claimed order of color inheritance, where the system of software components includes a lightweight peer component and a lightweight proxy component for displaying the child object with either the globally defined or inherited background color, if the background color of the child object is not declared.**

Claim 17 recites, "the system as recited in claim 1, wherein the system of software components comprises a lightweight peer component and a lightweight proxy component for displaying the child object with either the globally defined or inherited background color, if the background color of the child object is not declared." As such, claim 17 places an additional limitation on present claim 1 by introducing the software components (lightweight peer and proxy components) that enable the presently claimed order of color inheritance to be realized.

As noted above in the Issue 1 arguments of Group I claims, Magelang, JavaSPEC, JavaOne, and Bogdan each fail to disclose the presently claimed order of color inheritance, and furthermore, cannot be modified or combined to do so. M-SUN is not relied upon in the final Office Action for teaching the

presently claimed order of color inheritance. Though M-SUN mentions that a "*lightweight* component is one that 'borrows' the screen resource of an ancestor..." (M-SUN, page 2), M-SUN does not teach, suggest, or provide motivation for the presently claimed order of color inheritance.

Statements in the final Office Action rely on M-SUN for teaching the limitations of present claim 17. For example, the Examiner admits that MageLang, JavaSPEC, JavaOne, and Bogdan fail to "explicitly state a peer and proxy component for displaying the child object..." but suggests that "M-SUN teaches a method of implementing swing components similar to that of MageLang, JavaSPEC, JavaOne, and Bogdan, but further teaches a proxy component that associates an object with a graphics resource component, and further displays the object, in that the proxy component is the swing class (see page 2, paragraph 2), and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component, in that the peer component is the ancestor (see page 2, paragraph 2)." (Final Office Action, page 8). The Examiner further suggests that "[i]t would have been obvious to one of ordinary skill in the art, having the teachings of MageLang, JavaSPEC, JavaOne, Bogdan and M-SUN before him at the time the invention was made to modify the swing component interface of MageLang, JavaSPEC, JavaOne, and Bogdan to include the combinational properties as did m-SUN." (Final Office Action, page 8). The Applicant respectfully disagrees with the above Office Action statements, for at least the reasons set forth in more detail below.

First of all, present claim 17 does NOT recite limitations on "a proxy component that associates an object with a graphics resource component, and further displays the object... and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component," as suggested by the Examiner in the above Office Action statements. Though similar limitations may be found in a related application, the limitations cited by the Examiner do not appear in the presently claimed case. Therefore, the limitations cited by the Examiner are not relevant to the patentability of present claim 17.

In addition, and contrary to the Examiner's suggestion, the mere mention of "Swing classes" within the teachings of M-SUN does not and cannot provide teaching for the proxy component recited in present claims 17 and 19. For example, M-SUN provides absolutely no teaching or suggestion for a "Swing class" (i.e., the "proxy component" allegedly disclosed by M-SUN) capable of displaying a child object with either a globally defined or inherited background color, if the background color of the child object is not declared. In fact, there is no mention of background color or methods of color inheritance

within the teachings of M-SUN. Therefore, M-SUN cannot be used to provide teaching or suggestion for the presently claimed proxy component.

Furthermore, the mere mention of "ancestors" within the teachings of M-SUN does not and cannot provide teaching for the peer component recited in present claims 17 and 18. For example, M-SUN provides absolutely no teaching or suggestion for an "ancestor" (i.e., the "peer component" allegedly disclosed by M-SUN) capable of redirecting a method call (for getting the background color) to a lightweight proxy component, which as noted above, is capable of displaying a child object with either a globally defined or inherited background color. M-SUN fails to disclose such a proxy component, and therefore, cannot disclose a peer component capable of routing method calls to such a proxy component. As a consequence, M-SUN cannot be used to provide teaching or suggestion for the presently claimed peer component.

2. There is no motivation to modify or combine the cited art to teach or suggest the presently claimed peer and proxy components.

As noted above, none of the cited art teaches or suggests the presently claimed peer and proxy components. The Examiner admits to there being no teaching within MageLang, JavaSPEC, JavaOne, and Bogdan, and arguments have been provided above in support of the lack of teaching within M-SUN.

In addition to the lack of teaching, Applicants assert that the cited art cannot be modified or combined to teach or suggest the presently claimed peer and proxy components, since the cited art fails to suggest the desirability of doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify

the cited references can be found within the cited references to teach or suggest the aforementioned limitation of claim 17, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination, teaches all of the limitations of claim 17, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

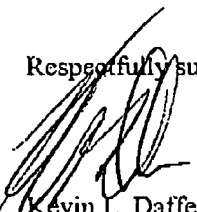
As explained in Arguments 1-3 above, at least some limitations of claim 17 and at least some limitations of claims 18-19, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 17-19 are patentably distinct over the cited art, and the rejection of Group II claims 17-19 under 35 U.S.C. § 103(a) is therefore asserted to be erroneous.

IX. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-6, 8-13, and 15-19 was erroneous, and reversal of the Examiner's decision is respectfully requested.

The Commissioner is hereby authorized to charge the required fee(s) to deposit account number 50-3268/5468-07700.

Respectfully submitted,



Kevin L. Daffer
Reg. No. 34,146
Attorney for Appellant

Daffer McDaniel, LLP
P.O. Box 684908
Austin, TX 78768-4908
Date: November 9, 2004

X. APPENDIX

The present claims on appeal are as follows.

1. A system of software components, which can be invoked during runtime by an application program running under an operating system to display a parent object and a child object, wherein the child object is displayed in accordance with the following background color inheritance order:

if a background color of the child object is declared, the child object is displayed with the declared background color;

if the background color of the child object is not declared, and a background color is globally defined for the system of components, the child object is displayed with the globally defined background color, which is independent of the operating system; and

if the background color of the child object is not declared, and the background color setting is not globally defined for the system of components, the child object is displayed with the background color of the parent object.

2. The system as recited in claim 1, wherein the parent and child objects are part of a graphical user interface associated with the application program.

3. The system as recited in claim 1, wherein the application program is written in Java programming language.

4. The system as recited in claim 1, wherein the system of software components comprises various software components within the Swing application program interface (API).

5. The system as recited in claim 1, wherein the operating system comprises a standard computer operating system such as Windows, Unix or OS/2.

6. The system as recited in claim 1, wherein the child object is one of multiple objects within a layout associated with the application program.

8. A method for color inheritance between a parent object and a child object, which are displayed by a system of software components created by an application program running on an operating system, wherein the method is adapted to display the child object in accordance with the following background color inheritance order:

if a background color of the child object is declared, displaying the child object with the declared background color;

if the background color of the child object is not declared, and a background color is globally defined for the system of components, displaying the child object with the globally defined background color, which is independent of the operating system; and

if the background color of the child object is not declared, and the background color setting is not globally defined for the system of components, displaying the child object with the background color of the parent.

9. The method as recited in claim 8, wherein the parent and child objects are part of a graphical user interface associated with the application program.

10. The method as recited in claim 8, wherein the application program is written in Java programming language.

11. The method as recited in claim 8, wherein the system of software components comprises various software components within the Swing application program interface (API).

12. The method as recited in claim 8, wherein the operating system comprises a Windows, Unix or OS/2 computer operating system.

13. The method as recited in claim 8, wherein the child object is one of multiple objects within a layout associated with the application program.

15. A computer-readable storage device, comprising:

a windows-based operating system;

a system of software components within the operating system;

a parent object and a child object, both created at runtime by an application running under the operating system, wherein the child object is displayed by the system of software components in accordance with the following background color inheritance order:

if a background color of the child object is declared when it is created, the child object is displayed with the declared background color;

if the background color of the child object is not declared when it is created, and a background color is globally defined for the system of components, the child object is displayed with the globally defined background color, which is independent of the operating system; and

if the background color of the child object is not declared when it is created, and the background color setting is not globally defined for the system of components, the child object is displayed with the background color of the parent.

16. The system as recited in claim 1, wherein the parent object is an AWT-based object or a Swing-based object.

17. The system as recited in claim 1, wherein the system of software components comprises a lightweight peer component and a lightweight proxy component for displaying the child object with either the globally defined or inherited background color, if the background color of the child object is not declared.

18. The system as recited in claim 17, wherein the lightweight peer component is adapted to redirect a method call for getting the background color to the lightweight proxy component.

19. The system as recited in claim 18, wherein the lightweight proxy component is adapted to translate the method call to determine whether the child object should be displayed using the globally defined or inherited background color.